Supervised Compression for Resource-Constrained Edge Computing Systems

Yoshitomo Matsubara  Ruihan Yang  Marco Levorato  Stephan Mandt
Department of Computer Science, University of California, Irvine
{yoshitom, ruihan.yang, levorato, mandt}@uci.edu

Abstract

There has been much interest in deploying deep learning algorithms on low-powered devices, including smartphones, drones, and medical sensors. However, full-scale deep neural networks are often too resource-intensive in terms of energy and storage. As a result, the bulk part of the machine learning operation is therefore often carried out on an edge server, where the data is compressed and transmitted. However, compressing data (such as images) leads to transmitting information irrelevant to the supervised task. Another popular approach is to split the deep network between the device and the server while compressing intermediate features. To date, however, such split computing strategies have barely outperformed the aforementioned naive data compression baselines due to their inefficient approaches to feature compression. This paper adopts ideas from knowledge distillation and neural image compression to compress intermediate feature representations more efficiently. Our supervised compression approach uses a teacher model and a student model with a stochastic bottleneck and learnable prior for entropy coding (Entropic Student). We compare our approach to various neural image and feature compression baselines in three vision tasks and found that it achieves better supervised rate-distortion performance while maintaining smaller end-to-end latency. We furthermore show that the learned feature representations can be tuned to serve multiple downstream tasks.

1. Introduction

With the abundance of smartphones, autonomous drones, and other intelligent devices, advanced computing systems for machine learning applications have become even more important [47, 10]. Machine learning models are frequently deployed on low-powered devices for reasons of computational efficiency or data privacy [45, 25]. However, deploying conventional computer vision or NLP models on such hardware raises a computational challenge, as powerful deep neural networks are often too energy-consuming to be deployed on such weak mobile devices [17].

An alternative to carrying out the deep learning model’s operation on the low-powered device is to send compressed data to an edge server that takes care of the heavy computation instead. However, recent neural or classical compression algorithms are either resource-intensive [48, 16] and/or optimized for perceptual quality [4, 38], and therefore much of the information transmitted is redundant for the machine learning task [13] (see Fig. 1). A better solution is to therefore split the neural network [28] into the two sequences so that some elementary feature transformations are applied by the first sequence of the model on the weak mobile (local) device. Then, intermediate, informative features are transmitted through a wireless communication channel to the edge server that processes the bulk part of the computation (the second sequence of the model) [18, 35].

Traditional split computing approaches transmit intermediate features by either reducing channels in convolution layers [28] or truncating them to a lower arithmetic precision [36, 46, 37]. Since the models were not “informed” about such truncation steps during training oftentimes leads to substantial performance degradation. This raises the question of whether learnable end-to-end data compression pipelines can be designed to both truncate and entropy-code the involved early-stage features.
In this paper, we propose such a neural feature compression approach by drawing on variational inference-based data compression [5, 48]. Our architecture resembles the variational information bottleneck objective [1] and relies on an encoder, a “prior” on the bottleneck state, and a decoder that leads to a supervised loss (see Fig. 1). At inference time, we discretize the encoder’s output and use our learned prior as a model for entropy coding intermediate features. The decoder reconstructs the feature vector losslessly from the binary bitstring and carries out the subsequent supervised machine learning task. Crucially, we combine this feature compression approach with knowledge distillation, where a teacher model provides the training data as well as parts of the trained architecture.\(^1\)

In more detail, our main contributions are as follows:

- We propose a new training objective for feature compression in split computing that allows us to use a learned entropy model for bottleneck quantization in conjunction with knowledge distillation.

- Our approach significantly outperforms seven strong baselines from the split computing and (neural) image compression literature in terms of rate-distortion performance (with distortion measuring a supervised error) and in terms of end-to-end latency.

- Moreover, we show that a single encoder network can serve multiple supervised tasks, including classification, object detection, and semantic segmentation.

2. Related Work

Neural Image Compression. Neural image compression methods apply neural networks for nonlinear dimensionality reduction and subsequent entropy coding. Early works [50, 27] leveraged LSTMs to model spatial correlations of the pixels within an image. The first proposed autoencoder architecture for image compression [49] used the straight-through estimator [8] for learning a discrete latent representation. The connection of image compression to probabilistic generative models was drawn by variational autoencoders (VAEs) [29, 4]. In the subsequent work [5], two-level VAE architectures involving a scale hyper-prior are proposed to encode images, which can be further improved by autoregressive structures [38, 39] or by optimization at encoding time [52]. Recent work also shows the potential progressive compression of the VAE structure by extending the quantization grid [33]. Other works [53, 19] demonstrate competitive image compression performance without a pre-defined quantization grid.

Recently, Dubois et al. [16] propose a self-supervised compression architecture for generic image classification. However, their encoder involves 87.8 million parameters (627 times larger than our encoder in Table 1) due to its Vision Transformer (ViT [15])-based encoder used in CLIP model [41] for ImageNet dataset. Thus, it does not satisfy resource-constrained edge computing systems.

Split Computing. Given that mobile (or local) devices often have limited resources such as computing power and battery, we usually transfer sensor data captured by the mobile device and offload heavy computing tasks to an edge (or cloud) server with more computing resources. Unlike local computing, which executes the entire model on the mobile device, edge computing (or full offloading) where the computation is on the edge server requires quality wireless communication between the mobile device and edge server. Otherwise, the total inference cost, such as the end-to-end latency, would be higher than local computing due to the communication delay, which would be critical for real-time applications. As an intermediate option between local computing and edge computing, split computing [28] has been attracting attention from research communities since edge computing is not always the best option. Specifically, the communication cost would be a severe problem for resource-limited edge computing systems [18, 35].

In split computing, a neural model will be split into the first and second sequences, and the first sequence of the model is executed on the mobile device. Having received the output of the first section via wireless communication, the second sequence of the model completes the inference on the edge server. A key concept is to reduce computational load on the mobile device while saving communication cost (data size) as processing delay on the edge server is often smaller compared to local processing and communication delays [37]. For reducing communication cost, recent studies on split computing [18, 35, 46, 37] introduce bottleneck, whose data size is smaller than input sample, to vision models. In recent studies, a combination of 1) fewer channels (channel reduction) in convolution layers and 2) quantization at bottleneck point is key to design such bottlenecks. While such studies show the effectiveness of their proposed approach for image classification and object detection tasks, the accuracy of bottleneck-injected models sharply drops when further compressing the bottleneck size as we will describe in Section 4.

Knowledge Distillation. It is widely known that deep neural models are often overparameterized [2, 51], and knowledge distillation [24] is one of the well-known techniques for model compression [9]. In the paradigm, a large pretrained model plays a role of teacher for a model to be trained (called student), and the student model learns from both hard-targets (e.g., one-hot vectors) and soft-targets (outputs of the teacher for the given input) during training.

\(^1\)Code and models are available at https://github.com/yoshitomo-matsubara/supervised-compression
Interestingly, some uncertainty from the pretrained teacher model as soft-target is informative to student models. The models trained with teachers often achieve better prediction performance than those trained without teachers [2]. As will be discussed later in this paper, learning compressed features for a target task such as image classification [48] would be challenging, specifically in the case that we introduce such bottlenecks to early layers in the model. We leverage a pretrained model as the teacher, and those with introduced bottlenecks as students to be trained to improve rate-distortion performance.

3. Method

After providing an overview of the setup (Section 3.1) we describe our distillation and feature compression approach (Section 3.2) and our procedure to fine-tune the model to other supervised downstream tasks (Section 3.3).

3.1. Overview

Our goal is to learn a lightweight, communication-efficient feature extractor for supervised downstream applications. We thereby transmit intermediate feature activations between two distinct portions of a neural network. The first part is deployed on a low-power mobile device, and the second part on a compute-capable edge server. Intermediate feature representations are compressed and transmitted between the mobile device and the edge server using discretization and subsequent lossless entropy coding.

In order to learn good compressible feature representations, we combine two ideas: knowledge distillation and neural data compression via learned entropy coding. First, we train a large teacher network on a data set of interest to teach a smaller student model. We assume that the features that the teacher model learns are helpful for other downstream tasks. Then, we train a lightweight student model to match the teacher model’s intermediate features (Section 3.2) with minimal performance loss. Finally, we fine-tune the student model to different downstream tasks (Section 3.3). Note that the training process is done offline.

The teacher network realizes a deterministic mapping \( x \mapsto h \mapsto y \), where \( x \) are the input data, \( y \) are the targets, and \( h \) are some intermediate feature representations of the teacher network. We assume that the teacher model is too large to be executed on the mobile device. The main idea is to replace the teacher model’s mapping \( x \mapsto h \) with a student model (i.e., the new targets become the teacher model’s intermediate feature activations). To facilitate the data transmission from the mobile device to the edge server, the student model, Entropic Student, embeds a bottleneck representation \( z \) that allows compression (see details below), and we transmit data as \( x \mapsto z \mapsto h \mapsto y \). We show that the student model can be fine-tuned to different tasks while the mobile device’s encoder part remains unchanged.

The whole pipeline is visualized in the bottom panel of Fig. 1. The latent representation \( z \) has a “prior” \( p(z) \), i.e., a density model over the latent space \( z \) that both sender and receiver can use for entropy coding after discretizing \( z \). In the following, we derive the details of the approach.

3.2. Knowledge Distillation

We first focus on the details of the distillation process. The entropic student model learns the mapping \( x \mapsto h \) (Fig. 2, left part) by drawing samples from the teacher model. The second part of the pipeline \( h \mapsto y \) (Fig. 2, right part) will be adapted from the teacher model and will be fine-tuned to different tasks (see Section 3.3).

Similar to neural image compression [4, 5], we draw on latent variable models whose latent states allow us to quantify and entropy-code data under a prior probability model. In contrast to neural image compression, our approach is supervised. As such, it mathematically resembles the deep Variational Information Bottleneck [1] (which was designed for adversarial robustness rather than compression).

**Distillation Objective.** We assume a stochastic encoder \( q(z|x) \), a decoder \( p(h|z) \), and a density model (“prior”) \( p(z) \) in the latent space. Specific choices are detailed below. Similar to [1], we maximize mutual information between \( z \) and \( h \) (making the compressed bottleneck state \( z \) as informative as possible about the supervised target \( h \)) while minimizing the mutual information between the input \( x \) and \( z \) (thus “compressing away” all the irrelevant information that does not immediately serve the supervised goal).

The objective for a given training pair \((x, h)\) provided by the teacher model is

\[
\mathcal{L}(x, h) = -\mathbb{E}_{q_{\phi}(z|x)} \left[ \log p_{\theta}(h|z) + \beta \log p_{\phi}(z) \right].
\]

Before discussing the rate and distortion terms, we specify and simplify this loss function further. Above, the decoder \( p(h|z) = \mathcal{N}(h; g_{\phi}(z), I) \) is chosen as a conditional Gaussian centered around a deterministic prediction \( g_{\phi}(z) \).
Input

\[ \text{Eq. 1 can be optimized via stochastic gradient descent and the prior distribution } p_{\phi} \text{ prediction representations from frozen layers besides } z \text{ the cross-entropy between the empirical distribution of } \]

In contrast, the second term measures the coding costs as \( \text{latent representation distortion tradeoff: the more aggressively we compress the } \)

\( \text{supervised data compression, our approach results in a rate-} \)

Similar to unsupervised data compression, our approach results in a rate-distortion tradeoff: the more aggressively we compress the \( \text{supervised target } \)

\( \text{learning} \) \[ \text{Equation 1 shows the base approach, describing the knowledge distillation pipeline with a single } h \text{ and involving } \]

\[ \text{a single target } y. \text{ In practice, our goal is to learn a compressed representation } z \text{ that does not only serve a single } \]

\( \text{supervised target } y, \text{ but multiple ones } y_1, \ldots, y_J. \text{ In particular, for a deployed system with a learned compression} \)

\( \text{module, we would like to be able to fine-tune the part of the network living on the edge server to multiple tasks without} \)

\( \text{having to retrain the compression model. As follows, we} \)

\( \text{show that such multi-task learning is possible.} \)

\( \text{A learned student model from knowledge distillation can be depicted as a two-step deterministic mapping } z = [f_\theta(x)] \text{ and } h = g_\phi(z), \text{ where } h (\approx h) \text{ is now a } \)

\( \text{decompressed intermediate hidden feature in our final student model (see Fig. 2). Assuming that } p_{\psi_j}(y_j|h) \text{ denotes the student model’s output probability distribution with parameters } \psi_j, \text{ the fine-tuning step amounts to optimizing} \)

\[ \psi_j^* = \arg \min_{\psi_j} - \mathbb{E}_{(x,y) \sim D} [p_{\psi_j}(y_j|g_\phi([f_\theta(x)]))]. \] (3)

The pair \( (y_j, \psi_j) \) refers to the target label and the parameters of each downstream task. The formula illustrates the Maximum Likelihood Estimation (MLE) method to optimize the parameter \( \psi_j \) for task \( j \). Note that we optimize the discriminative model after the compression model is frozen, so \( \theta \) is fixed in this training stage, and \( \phi \) can either be fixed or trainable. We elucidate the hybrid model in Fig. 2.

\( \text{theory [14]. The tradeoff is determined by the Lagrange multiplier } \beta. \)

\( \text{As a particular instantiation of an information bottleneck framework [1], Singh et al. [48] proposed a similar loss function as Eq. 1 to train a classifier with a bottleneck at its penultimate layer without knowledge distillation. In Section} \)

\( \text{4, we compare against a version of this approach that is compatible with our architecture and find that the knowledge distillation aspect is crucial to improve performance.} \)

\( \text{3.3. Fine-tuning for Target Tasks} \)

The first term in Eq. 1 measures the supervised distortion, as it expresses the average prediction error under the coding procedure of first mapping \( x \) to \( z \) and then \( z \) to \( h. \) In contrast, the second term measures the coding costs as the cross-entropy between the empirical distribution of \( z_i \) and the prior distribution \( p_\phi(z) \) according to information

\( \text{theory [14]. The tradeoff is determined by the Lagrange multiplier } \beta. \)

\( \text{As a particular instantiation of an information bottleneck framework [1], Singh et al. [48] proposed a similar loss function as Eq. 1 to train a classifier with a bottleneck at its penultimate layer without knowledge distillation. In Section} \)

\( \text{4, we compare against a version of this approach that is compatible with our architecture and find that the knowledge distillation aspect is crucial to improve performance.} \)

\( \text{3.3. Fine-tuning for Target Tasks} \)

The first term in Eq. 1 measures the supervised distortion, as it expresses the average prediction error under the coding procedure of first mapping \( x \) to \( z \) and then \( z \) to \( h. \) In contrast, the second term measures the coding costs as the cross-entropy between the empirical distribution of \( z_i \) and the prior distribution \( p_\phi(z) \) according to information

\( \text{theory [14]. The tradeoff is determined by the Lagrange multiplier } \beta. \)

\( \text{As a particular instantiation of an information bottleneck framework [1], Singh et al. [48] proposed a similar loss function as Eq. 1 to train a classifier with a bottleneck at its penultimate layer without knowledge distillation. In Section} \)

\( \text{4, we compare against a version of this approach that is compatible with our architecture and find that the knowledge distillation aspect is crucial to improve performance.} \)

\( \text{3.3. Fine-tuning for Target Tasks} \)

The first term in Eq. 1 measures the supervised distortion, as it expresses the average prediction error under the coding procedure of first mapping \( x \) to \( z \) and then \( z \) to \( h. \) In contrast, the second term measures the coding costs as the cross-entropy between the empirical distribution of \( z_i \) and the prior distribution \( p_\phi(z) \) according to information
For fine-tuning the student model with the frozen encoder, we leverage a teacher model again. For image classification, we apply a standard knowledge distillation technique [24] to achieve better model accuracy by distilling the knowledge in the teacher model into our student model. Specifically, we fine-tune the student model by minimizing a weighted sum of two losses: 1) cross-entropy loss between the student model’s class probability distribution and one-hot vector (hard-target), and 2) Kullback-Leibler divergence between softened class probability distributions from both the student and teacher models.

Similarly, having frozen the encoder, we can fine-tune different models for different downstream tasks reusing the trained entropic student model (classifier) as their backbone, which will be demonstrated in Section 4.4.

4. Experiments

Using torchdistill [34], we designed different experiments and studied various models based on both principles of split-computing (partial offloading) and edge computing (full offloading). We used ResNet-50 [23] as a base model, which, besides image classification, is also widely used as a backbone for different vision tasks such as object detection [22, 31] and semantic segmentation [12]. In all experiments, we empirically show that our approach leads to better supervised rate-distortion performance.

4.1. Baselines

In this study, we use seven baseline methods categorized into either input compression or feature compression.

**Input compression (IC).** A conventional implementation of the edge computing paradigm is to transmit the compressed image directly to the edge server, where all the tasks are then executed. We consider five baselines referring to this “input compression” scenario: JPEG, WebP [20], BPG [7], and two neural image compression methods (factorized prior and mean-scale hyperprior) [5, 38] based on CompressAI [6]. The latter approach is currently considered state of the art in image compression models (without autoregressive structure) [38, 39, 52]. We evaluate each model’s performance in terms of the rate-distortion curve by setting different quality values for JPEG, WebP, and BPG and Lagrange multiplier $\beta$ for neural image compression.

**Feature compression (FC).** Split computing baselines [36, 46] correspond to reducing the bottleneck data size with channel reduction and bottleneck quantization referred to as CR+BQ (quantizes 32-bit floating-point to 8-bit integer) [26]. Matsubara et al. [37, 36] report that bottleneck quantization did not lead to significant accuracy loss. To control the rate-distortion tradeoff, we design bottlenecks with a different number of output channels in a convolution layer to control the bottleneck data size, train the bottleneck-injected models and quantize the bottleneck after the training session.

Our final baseline in this paper is an end-to-end approach towards learning compressible features for a single task similar to Singh et al. [48] (for brevity, we will cite their reference). Their originally proposed approach focuses only on classification and introduces the compressible bottleneck to the penultimate layer. In the considered setting, such design leads to an overwhelming workload for the mobile/local device: for example, in terms of model parameters, about 92% of the ResNet-50 [23] parameters would be deployed on the weaker, mobile device. To make this approach compatible with our setting, we apply their approach to our architecture; that is, we directly train our entropic student model without a teacher model. We find that compared to [48], having a stochastic bottleneck at an earlier layer (due to limited capacity of mobile devices) leads to a model that is much harder to optimize (see Section 4.3).

4.2. Implementation of Our Entropic Student

Vision models in recent years reuse pretrained image classification models as their backbones e.g., ResNet-50 [23] as a backbone of RetinaNet [31] and Faster R-CNN [42] for object detection tasks. These models often use intermediate hidden features extracted from multiple layers in the backbone as the input to subsequent task-specific modules such as feature pyramid network (FPN) [30]. Thus, using an architecture with a bottleneck introduced at late layers [48] for tasks other than image classification may require transferring and compressing multiple hidden features to an edge server, which will result in high communication costs.

To improve the efficiency of split computing compared to that of edge computing, we introduce the bottleneck as early in the model as possible to reduce the computational workload at the mobile device. We replace the first layers of our pretrained teacher model with the new modules for encoding and decoding transforms as illustrated in Fig. 3. The student model, entropic student, consists of the new modules and the remaining layers copied from its teacher model for initialization. Similar to neural image compression models [5, 38], we use convolution layers and simplified generalized divisive normalization (GDN) [3] layers to design an encoder $f_\theta$, and design a decoder $g_\phi$ with convolution and inverse version of simplified GDN (IGDN) layers. Importantly, the designed encoder should be lightweight, e.g., with fewer model parameters as it will be deployed and executed on a low-powered mobile device. We will discuss the deployment cost in Section 4.6.

Different from bottleneck designs in the prior studies on split computing [37, 36], we control the trade-off between bottleneck data size and model accuracy with the $\beta$ value in the rate-distortion loss function (See Eq. 2).
4.3. Image Classification

We first discuss the rate-distortion performance of our and baseline models using a large-scale image classification dataset. Specifically, we use ImageNet (ILSVRC 2012) [43], that consists of 1.28 million training and 50,000 validation samples. As is standard, we train the models on the training split and report the top-1 accuracy on the validation split. Using ResNet-50 [23] pre-trained on ImageNet as a teacher model, we replace all the layers before its second residual block with our encoder and decoder to compose our entropic student model. The introduced encoder-decoder modules are trained to approximate model. The introduced encoder-decoder entropic student residual block with our encoder and decoder to compose our a teacher model, we replace all the layers before its second split. Using ResNet-50 [23] pre-trained on ImageNet as the training split and report the top-1 accuracy on the validation samples. As is standard, we train the models on 2012) [43], that consists of 1.28 million training and 50,000 and baseline models using a large-scale image classification task. A popular approach used in split computing studies, the combination of channel reduction and bottleneck quantization (CR+BQ) [37], seems slightly better than JPEG compression but not as accurate as those with the neural image compression models.

Among all the configurations in the figure, our model trained by the two-stage method performs the best. We also trained our model without teacher model, which in essence corresponds to [48]. The resulting RD curve is significantly worse, which we attribute to two possible effects: first, it is widely acknowledged that knowledge distillation generally finds solutions that generalize better. Second, having a stochastic bottleneck at an earlier layer may make it difficult for the end-to-end training approach to optimize.

4.4. Object Detection and Semantic Segmentation

As suggested by He et al. [21], image classifiers pre-trained on the ImageNet dataset [43] speed up the convergence of training on downstream tasks. Reusing the proposed model pre-trained on the ImageNet dataset, we further discuss the rate-distortion performance on two downstream tasks: object detection and semantic segmentation. Specifically, we train RetinaNet [31] and DeepLabv3 [12], using our models pre-trained on the ImageNet dataset in the previous section as their backbone. RetinaNet is a one-stage object detection model that enables faster inference than two-stage detectors such as Mask R-CNN [22]. DeepLabv3 is a semantic segmentation model that leverages Atrous Spatial Pyramid Pooling (ASPP) [11].

For the downstream tasks, we use the COCO 2017 dataset [32] to fine-tune the models. The training and validation splits in the COCO 2017 dataset have 118,287 and 5,000 annotated images, respectively. As detection performance, we refer to mean average precision (mAP) for...
Bounding box (BBox) outputs with different Intersection-over-Unions (IoU) thresholds from 0.5 and 0.95 on the validation split. For semantic segmentation, we measure the performance by pixel IoU averaged over 21 classes present in the PASCAL VOC 2012 dataset. It is worth noting that following the PyTorch [40] implementations, the input image scales for RetinaNet [31] are defined by the shorter image side and set to 800 in this study which is much larger than the input image in the previous image classification task. As for DeepLabv3 [12], we use the resized input images such that their shorter size is 520. The training setup and hyperparameters used to fine-tune the models are described in the supplementary material.

Similar to the previous experiment for the image classification task, Figures 5 and 6 show that the combinations of neural compression models and the pre-trained RetinaNet and DeepLabv3, which are still strong baselines in object detection and semantic segmentation tasks. Our model demonstrates better rate-distortion curves in both tasks. In the object detection task, our model’s improvements over RetinaNet with BPG and mean-scale hyperprior are smaller than those in the image classification and semantic segmentation tasks. However, our model’s encoder to be executed on a mobile device is approximately 40 times smaller than the encoder of the mean-scale hyperprior. Our model also can achieve a much shorter latency to complete the input-to-prediction pipeline (see Fig. 1) than the baselines we considered for resource-constrained edge computing systems. We further discuss these aspects in Sections 4.6 and 4.7.

4.5. Bitrate Allocation of Latent Representations

This section discusses the difference between the representations of bottlenecks in neural image compression and our models. We are interested in which element of the bottlenecks allocates more bits in the latent representation \( z \). Bottlenecks in neural image compression models will allocate many bits to some unique area in an image to preserve all its characteristics in the reconstructed image. On the other hand, those in our models are trained to mimic the feature representations in their teacher model, thus expected to allocate more bits to areas useful for the target task.

Figure 7 shows visualizations of the normalized bitrate allocations for a few sample images. The 2nd column of the figure corresponds to the bottleneck in a neural image compression model prioritizing the images’ backgrounds such as catcher’s zone and glasses. Interestingly, our bottleneck representation (the 3rd column) seems to eliminate the difference between the two backgrounds and focuses on objects in the images such as persons and soccer ball. Moreover, the bottleneck eliminates a digital watermark at the top left in the first image, which is most likely not critical for the target task, while the one in the neural compression model noticeably distinguishes the logo from the background.
data to complete the inference task with a full classification model. Thus, only the compressor in the input compression model is accounted for in the computation cost on the mobile device. In Section 4, the two input compression models, factorized prior [5] and mean-scale hyperprior [38], are strong baseline approaches, and mean-scale hyperprior outperforms the factorized prior in terms of rate-distortion curve. However, its model size is comparable to or more expensive than popular lightweight models such as MobileNetV2 [45] and MobileNetV3 [25]. For this reason, this strategy is not advantageous unless the model deployed on the edge server can offer much higher accuracy than the lightweight models on the mobile device.

In contrast, split computing (SC) models, including our entropic student model, perform in-network feature compression while extracting features from the target task’s input sample. As shown in Table 1, the encoder of our model is much smaller (about 10 – 40 times smaller than) compared to those of the input compression models and the lightweight classifiers. Moreover, the encoder in our student model can be shared with RetinaNet and DeepLabv3 for different tasks. When a mobile device has multiple tasks such as image classification, object detection, and semantic segmentation, the single encoder is on memory and executed for an input sample. We note that ResNet-50 models with channel reduction and bottleneck quantization [37] and those for compressive feature [48] in their studies require a non-shareable encoder for different tasks. With their approaches, there are three individual encoders on the memory of the more constrained mobile device, which leads to approximately 3 times larger deployment cost.

4.7. End-to-End Prediction Latency Evaluation

To compare the prediction latency with the different approaches, we deploy the encoders on two different mobile devices: Raspberry Pi 4 (RPI4) and NVIDIA Jetson TX2 (JTX2). As an edge server (ES), we use a desktop computer with an NVIDIA GeForce RTX 2080 Ti, assuming the use of LoRa [44] for low-power communications (maximum data rate is 37.5 Kbps). For all the considered approaches, we use the data points with about 74% accuracy in Fig. 4, and the end-to-end latency is the sum of 1) execution time to encode an input image on RPI4/JTX2, 2) delay to transfer the encoded data from RPI4/JTX2 to ES, and 3) execution time to decode the compressed data and complete inference on ES.

Table 2 shows our approach reduces the end-to-end prediction latency by 47 – 62% compared to the baselines. The encoding time and communication delay are dominant in the end-to-end latency while the execution time on ES is negligible. For both the experimental configurations (RPI4 → ES and JTX2 → ES), the breakdowns of the end-to-end latency are illustrated in the supplementary material.

Table 2: End-to-end latency to complete input-to-prediction pipeline for resource-constrained edge computing systems illustrated in Fig. 1, using RPI4/JTX2, LoRa and ES. The breakdowns are available in the supplementary material.

<table>
<thead>
<tr>
<th>Approach</th>
<th>RPI4 → ES</th>
<th>JTX2 → ES</th>
</tr>
</thead>
<tbody>
<tr>
<td>JPEG + ResNet-50</td>
<td>2.35 sec</td>
<td>2.34 sec</td>
</tr>
<tr>
<td>WebP + ResNet-50</td>
<td>1.83 sec</td>
<td>1.84 sec</td>
</tr>
<tr>
<td>BPG + ResNet-50</td>
<td>2.46 sec</td>
<td>2.41 sec</td>
</tr>
<tr>
<td>Factorized Prior + ResNet-50</td>
<td>2.43 sec</td>
<td>2.22 sec</td>
</tr>
<tr>
<td>Mean-Scale Hyperprior + ResNet-50</td>
<td>2.24 sec</td>
<td>1.92 sec</td>
</tr>
<tr>
<td>ResNet-50 w/ BQ</td>
<td>2.27 sec</td>
<td>2.25 sec</td>
</tr>
<tr>
<td>Our Entropic Student</td>
<td>0.972 sec</td>
<td>0.904 sec</td>
</tr>
</tbody>
</table>

5. Conclusions

This paper adopts ideas from knowledge distillation and neural image compression to achieve feature compression for supervised tasks. Our approach leverages a teacher model to introduce a stochastic bottleneck and a learnable prior for entropy coding at its early stage of a student model (namely, Entropic Student). The framework reduces the computational burden on the weak mobile device by offloading most of the computation to a computationally powerful cloud/edge server, and the single encoder in our entropic student can serve multiple downstream tasks. The experimental results show the improved supervised rate-distortion performance for three different vision tasks and the shortened end-to-end prediction latency, compared to various (neural) image compression and feature compression baselines.

To ensure reproducibility of the experimental results and facilitate research to address this important problem, we release the training code and trained models at https://github.com/yoshitomo-matsubara/supervised-compression.

Acknowledgements

This material is in part based upon work supported by the Defense Advanced Research Projects Agency (DARPA) under Contract No. HR001120C0021. Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of DARPA. We acknowledge the support by the Google Cloud Platform research credits, and the National Science Foundation under the NSF CAREER award 2047418 and Grants 1928718, 2003237, 2007719, IIS-1724331 and MLWiNS-2003237, as well as Intel and Qualcomm. We also thank Yibo Yang and the anonymous reviewers for their insightful comments.
References


[28] Yiping Kang, Johann Hauswald, Cao Gao, Austin Rovinski, Trevor Mudge, Jason Mars, and Lingjia Tang. Neurosurgeon: Collaborative Intelligence Between the Cloud and Mobile Edge. In Proceedings of the Twenty-Second Intern-


